

Monotony in Service Orchestrations

Anne Bouillard — Sidney Rosario — Albert Benveniste — Stefan Haar

N° 6528

May 2008

Thème COM

 *apport
de recherche*

Monotony in Service Orchestrations ^{*}

Anne Bouillard, Sidney Rosario, Albert Benveniste , Stefan Haar [†]

Thème COM — Systèmes communicants
Équipes-Projets Distribcom

Rapport de recherche n° 6528 — May 2008 — 20 pages

Abstract: Web Service orchestrations are compositions of different Web Services to form a new service. The services called during the orchestration guarantee a given performance to the orchestrator, usually in the form of contracts. These contracts can be used by the orchestrator to deduce the contract it can offer to its own clients, by performing contract composition. An implicit assumption in contract based QoS management is: "the better the component services perform, the better the orchestration's performance will be". Thus, contract based QoS management for Web services orchestrations implicitly assumes monotony.

In some orchestrations, however, monotony can be violated, i.e., the performance of the orchestration improves when the performance of a component service degrades. This is highly undesirable since it can render the process of contract composition inconsistent.

In this paper we define monotony for orchestrations modelled by Colored Occurrence Nets (CO-nets) and we characterize the classes of monotonic orchestrations. We show that few orchestrations are indeed monotonic, mostly since latency can be traded for quality of data. We also propose a sound refinement of monotony, called *conditional monotony*, which forbids this kind of cheating and show that conditional monotony is widely satisfied by orchestrations. This finding leads to reconsidering the way SLAs should be formulated.

Key-words: web service, orchestrations, contracts, monotony

^{*} This work was partially funded by the ANR national research program DOTS (ANR-06-SETI-003), DocFlow (ANR-06-MDCA-005) and the project CREATE ActivDoc.

[†] S.Rosario and A. Benveniste are with Irisa/Inria, Campus de Beaulieu, Rennes. A. Bouillard is with Irisa/ENS Cachan Campus de Ker Lann and S.Haar is with Alcatel-Lucent Bell Labs, Kanata, ON, Canada.

Monotonie dans les orchestrations de web services

Résumé : Les orchestrations de services web sont des compositions de services élémentaires. Ces services, fournissent un 'contrat' à l'orchestrateur, ce qui garantit une certaine performance de leur service. Ces contrats sont utilisés par l'orchestrateur pour proposer un contrat à un client pour son propre service. Cela se fait par la 'composition de contrats'. Du point de vue de la performance, la composition de contrats suppose implicitement que "L'amélioration de la performance d'un service va rendre l'orchestration plus performante". La composition de contrats suppose ainsi que les orchestrations sont "monotones".

Dans quelques orchestrations, cependant, la monotonie peut ne pas être respectée. Lorsque la performance d'un service s'améliore, la performance de l'orchestration se dégrade. Ceci est très gênant car cela rend le processus de composition de contrats invalide. Dans ce rapport, nous définissons la monotonie pour les orchestrations modélisées par des réseaux d'occurrence colorés (CO-nets) et nous caractérisons la classe des orchestrations monotones. Nous démontrons que très peu d'orchestrations sont monotone en pratique, ce qui est largement dû à la possibilité d'améliorer la latence en dégradant la qualité de la réponse donnée. Nous proposons ensuite un raffinement de la monotonie, la "monotonie conditionnelle", qui interdit ce type de 'triche'. Nous montrons que la monotonie conditionnelle est très généralement satisfaite par les orchestrations. Cette étude nous mène à reconsidérer la formulation des contrats dans le cadre des orchestrations de services web.

Mots-clés : web service, orchestrations, contrats, monotonie

Contents

1	Introduction	3
2	Non-monotonic patterns in <i>CO</i>-nets	5
3	The Orchestration Model: OrchNets	7
3.1	Petri nets, Occurrence nets	7
3.2	Our Model: OrchNets	9
4	Characterizing monotony	12
4.1	Defining monotony	12
4.2	A global necessary and sufficient condition	13
4.3	A structural condition for the monotony of workflow nets	14
4.4	Discussion regarding monotony	17
5	Refined QoS and Conditional monotony	18
6	Conclusion	19

1 Introduction

Web Services and their compositions are being widely used to build distributed applications over the internet. Web Service orchestrations are compositions of Web Services to form an aggregate, and usually more complex, Web Service (WS). The services involved in a WS orchestration may have widely different behaviour (both functional and non-functional) and may be separated geographically by large distances.

A WS orchestration is itself a Web Service, and it can be further composed with other Web Services to build increasingly sophisticated services. The functioning of such service compositions can thus be quite complex in nature. There is a need to formally describe these systems in order to be able to build and reason about them. Different formalisms have been proposed for this purpose, the most popular amongst these is the Business Process Execution Language (BPEL) [2] a standard proposed by Microsoft and IBM. Another formalism is Orc [7] a small and elegant formalism equipped with extensive semantics work [5, 10]. Various models exist, that have been either used to model directly orchestrations or choreographies, or as a semantic domain for some formalisms. Noticeably Petri Nets based models, e.g., WorkFlow Nets [1, 11] and process algebra based models.

The main focus of the existing models is to capture the functional aspects of such compositions. However, non-functional — also called Quality of Service (QoS) — aspects involved in services and their compositions need also to be considered. The QoS of a service is characterised by different metrics, for *e.g.*, latency, availability, throughput, security, etc. Standard WSLA [4] specifies how QoS can be specified using Service Level Specifications and Agreements. Examples of SLA parameters are found in this document that illustrate the current practice. In particular, “conditional SLA obligations” consist in specifying what a service must guarantee given that the environment satisfies certain assumptions.

QoS management is typically based on the notion of *contract*. Contracts are agreements made between the orchestrator and the different actors (the called services) involved in the orchestration. Contracts formalise the duties and responsibilities each subcontractor must satisfy. For *e.g.*, a service which is called in an orchestration can have a contract of the type : *for 95% of the requests the response time will be less than 5ms*. All these contracts with the services involved in the orchestration could then be composed by the orchestrator to help it propose its own contract with users of the orchestration. This process is called *contract composition*.

In [9] we introduced the notion of *probabilistic contracts* to formalise the QoS behaviour of services — the work of [9] focused on latency. We showed how these contracts can be composed to get the end-to-end orchestration's contract. We also showed that realistic *overbooking* of resources by the orchestrator is possible using this approach.

Contract based QoS management relies on the implicit assumption that, if each sub-contractor meets its contract objectives, then so does the orchestrator. Vice-versa, a sub-contractor breaching its contract can cause the orchestrator to fail meeting its overall contract objectives. Thus the whole philosophy behind contracts is that the better sub-contractors behave, the better the overall orchestration will meet its own contract. In fact, the authors themselves have developed their past work [9] based on this credo... until they discovered that this implicit assumption could easily be falsified. Why so?

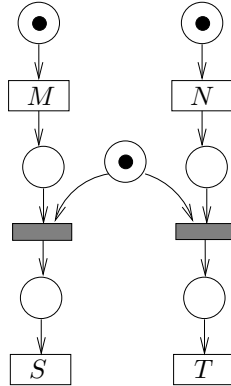


Figure 1: A non-monotonic orchestration

As an illustration example, consider the orchestration in Figure 1. Services M and N are first called in parallel. If M responds first, service S is next called and the response of N is ignored. If N responds first, T is called and not S . Let δ_i denote the response time of site i . Assume the following delay behaviour : $\delta_M < \delta_N$ and $\delta_S \gg \delta_T$. Since M responds faster, the end-to-end orchestration delay $d_0 = \delta_M + \delta_S$. Now let service M behaves slightly 'badly', *i.e.* delay δ_M increases and becomes slightly greater than δ_N . Now service T is called and the new orchestration delay is $d_1 = \delta_N + \delta_T$. But since $\delta_S \gg \delta_T$, d_1 is in fact lesser than d_0 . This orchestration is non-monotonic since increasing the latency of one of its components can decrease the end-to-end latency of the orchestration. So, what is the nature of the difficulty?

“Simple” composed Web services are such that QoS aspects do not interfere with functional aspects and do not interfere with each other. Their flow of control is typically rigid and does not involve if-then-else branches. For such cases, latencies will compose gently and will not cause pathologies as shown above. However, as evidenced by the rich constructions offered by BPEL, orchestrations and choreographies can have branching based on data and QoS values, various kinds of exceptions, and timers. With such flexibility, non-monotony such as that exhibited by the example of Figure 1 can very easily occur.

Lack of monotony, in turn, impairs using contracts for the compositional management of QoS. Surprisingly enough, this fact does not seem to have been noticed in the literature.

In this paper we give a classification for orchestrations based on their monotonic characteristics. Our study focuses on latency, although other aspects of QoS are discussed as well. This paper is organised as follows: Section 2 informally introduces the notion of monotony with examples. In section 3 we recall the definition of Petri nets and colored Petri nets, and introduce our model, OrchNet. A formal definition of monotony and a characterisation of monotonic orchestrations is then given in section 4. Section 5 introduces *conditional* monotony which will be useful to deal with non-monotonic orchestrations which use data-dependent control.

2 Non-monotonic patterns in CO-nets

We now show examples of non-monotonic orchestrations and identify the source of their non-monotony. The necessary and sufficient conditions for monotony that we give later were inspired from the study of these patterns. The examples we discuss here informally use Petri nets; we believe they are self-explanatory and do not deserve formal definitions. The formal definitions of Petri nets and their semantics is given in section 3.

Consider the net on Figure 2 with four transitions a, b, c and d with latencies τ_a, τ_b, τ_c and τ_d respectively. Let $\tau_a = 2$, $\tau_b = 3$, $\tau_c = 4$ and $\tau_d = 7$. Here a fires before b and the overall orchestration latency is $\tau_a + \tau_c = 6$. Now, if $\tau_b = 1$, b fires before a and so the orchestration latency is $\tau_b + \tau_d = 8$. Since decreasing the latency of b increases the orchestration’s latency, the net is not monotonic. Non-monotony arises from the fact that the futures of the conflicting transitions a and b had different latencies. The net is monotonic if the latencies of c and d are constrained to be the same.

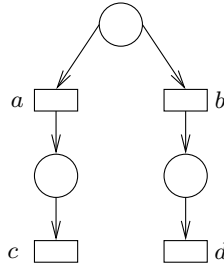


Figure 2: Choices are not monotonic.

In Figure 3, transitions a and c are concurrent, $a\#b$ and $b\#c$. Since we saw in the previous example that the futures of conflicting transitions should have the same latencies, let us set $\tau_d = \tau_e = \tau_f = \tau_*$. If $\tau_a = 4, \tau_b = 3, \tau_c = 5$, b fires first and the overall latency is $\tau_b + \tau_e = 4 + \tau_*$. Now if $\tau_a = 2$, a fires first, b is blocked and c will eventually fire. The overall latency here is $\max(\tau_a + \tau_d, \tau_c + \tau_f) = 5 + \tau_*$. We thus see that for this net to be monotonic, we must have the latencies of the concurrent transitions a and c to be the same (along with their futures). In fact, since $\tau_a = \tau_c$ and $\tau_d = \tau_f$, the two concurrent branches can be "folded" into a single branch to give us a net similar to that of Figure 2.

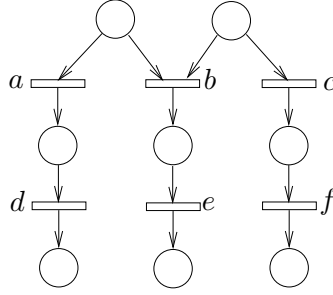


Figure 3: Choice and Concurrency inter-playing

A monotonic orchestration: Figure 4 shows an orchestration that has a fork-join pattern (in the left). The right side of the figure shows the net's *unfolding*. (The unfolding of a net N is an acyclic net which includes all the possible executions of N . Every place in the unfolding can be marked by at-most one transition and so every possible way of enabling a transition is distinguished. In Figure 4, the two possible cases in which event c can be fired are distinguished in the unfolded net). Since the left net has a choice pattern, one may think that it is not monotonic. However, this is not true. From the previous example we know that the unfolded net is monotonic since the futures of the conflicting transitions a and b are the same (and hence have the same latencies).

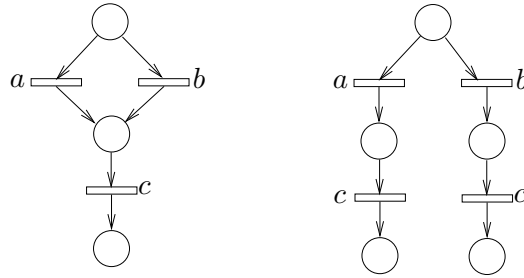


Figure 4: A monotonic orchestration with a fork-join pattern (left) and its unfolding (right).

At this point we have identified good and bad patterns for monotony. Going beyond such simple findings proved to be surprisingly challenging. In particular,

characterizing that the restricted class of orchestrations we define is a *necessity* for monotony is demanding. All this requires more formal material that we introduce next.

3 The Orchestration Model: OrchNets

In this section we present the orchestration model that we use for our studies, which we call *OrchNets*. OrchNets are a special form of *colored occurrence nets* (*CO-nets*), which are high level Petri Nets.

We have chosen this mathematical model for the following reasons. From the semantic studies performed for BPEL [8, 3] and Orc [5, 10] we need to support in an elegant and succinct way the following features: concurrency, rich control patterns including preemption, and for some cases even recursion. The first two requirements suggest using colored Petri nets. The last requirement suggests considering extensions of Petri nets with dynamicity. However, in our study we will not be interested in the specification of orchestrations, but rather in their executions. Occurrence nets are concurrent models of executions of Petri nets. As such, they encompass orchestrations involving recursion at no additional cost. The executions of Workflow Nets [1, 11] are also (CO-nets).

3.1 Petri nets, Occurrence nets

Definition 1 A Petri net is a tuple $N = (\mathcal{P}, \mathcal{T}, \mathcal{F}, M_0)$, where

- \mathcal{P} is a set of places,
- \mathcal{T} is a set of transitions such that $\mathcal{P} \cap \mathcal{T} = \emptyset$,
- $\mathcal{F} \subseteq (\mathcal{P} \times \mathcal{T}) \cup (\mathcal{T} \times \mathcal{P})$ is a set of flow arcs,
- $M_0 : \mathcal{P} \rightarrow \mathbf{N}$ is the initial marking.

The elements in $\mathcal{P} \cup \mathcal{T}$ are called the *nodes* of N and will be denoted by variables for e.g. x . For node $x \in \mathcal{P} \cup \mathcal{T}$, we call $\bullet x = \{y \mid (y, x) \in \mathcal{F}\}$ the *preset* of x , and $x^\bullet = \{y \mid (x, y) \in \mathcal{F}\}$ the *postset* of x . A *marking* of the net is a multiset M of places, i.e a map from \mathcal{P} to \mathbf{N} . A transition t is *enabled* in marking M if $\forall p \in \bullet t, M(p) > 0$. This enabled transition can *fire* resulting in a new marking $M - \bullet t + t^\bullet$ denoted by $M[t]M'$. A marking M is *reachable* if there exists a sequence of transitions $t_0, t_1 \dots t_n$ such that $M_0[t_0]M_1[t_1] \dots [t_n]M$. A net is *safe* if for all reachable markings M , $M(p) \subseteq \{0, 1\}$ for all $p \in \mathcal{P}$. We define two relations on the nodes of a net:

Definition 2 (Causality) For a net $N = (\mathcal{P}, \mathcal{T}, \mathcal{F}, M_0)$ the causality relation $<$ is the transitive closure of the relation \prec defined as:

- If $p \in \bullet t$, then $p \prec t$,
- If $p \in t^\bullet$, then $t \prec p$,

where $p \in \mathcal{P}, t \in \mathcal{T}$.

The reflexive closure of $<$ is denoted by \leq . For a node $x \in \mathcal{P} \cup \mathcal{T}$, the set of *causes* of x is $\lceil x \rceil = \{y \in \mathcal{P} \cup \mathcal{T} \mid y \leq x\}$.

Definition 3 (Conflict) For a net $N = (\mathcal{P}, \mathcal{T}, \mathcal{F}, M_0)$ two nodes x and y are in conflict - denoted by $x \# y$ - if there exist distinct transitions $t, t' \in \mathcal{T}$, such that $t \leq x, t' \leq y$ and $\bullet t \cap \bullet t' \neq \emptyset$.

Nodes x and y are said to be *concurrent* - written as $x \parallel y$ - if neither $(x \leq y)$ nor $(y \leq x)$ nor $(x \# y)$. A set of concurrent conditions $P \subseteq \mathcal{P}$ is called a *co-set*. A *cut* is a maximal (for set inclusion) co-set.

Definition 4 (Configuration) A configuration of N is a subnet κ of nodes of N such that:

1. κ is causally closed, i.e, if $x < x'$ and $x' \in \kappa$ then $x \in \kappa$
2. κ is conflict-free, i.e, for all nodes $x, x' \in \kappa$, $\neg(x \# x')$

For convenience, we will assume that the maximal nodes (w.r.t the $<$ relation) in a configuration are places.

Definition 5 (Occurrence nets) A safe net $N = (\mathcal{P}, \mathcal{T}, \mathcal{F}, M_0)$ is called an occurrence net (O-net) iff

1. $\neg(x \# x)$ for every $x \in \mathcal{P} \cup \mathcal{T}$.
2. \leq is a partial order and $[t]$ is finite for any $t \in \mathcal{T}$.
3. For each place $p \in \mathcal{P}$, $|\bullet p| \leq 1$.
4. $M_0 = \{p \in \mathcal{P} \mid \bullet p = \emptyset\}$, i.e the initial marking is the set of minimal places with respect to \leq_N .

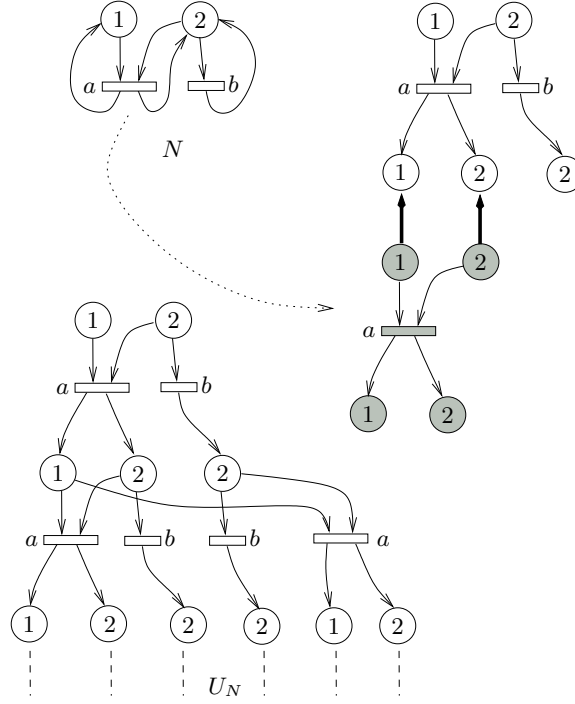
Occurrence nets are a good model for representing the possible executions of a concurrent system. *Unfoldings* of a safe Petri net, which collect all the possible executions of the net, are occurrence nets. Unfoldings are defined as follows.

For N and N' two safe nets, a map $\varphi : \mathcal{P} \cup \mathcal{T} \mapsto \mathcal{P}' \cup \mathcal{T}'$ is called a *morphism* of N to N' if: 1/ $\varphi(\mathcal{P}) \subseteq \mathcal{P}'$ and $\varphi(\mathcal{T}) \subseteq \mathcal{T}'$, and 2/ for every $t \in \mathcal{T}$ and $t' = \varphi(t) \in \mathcal{T}'$, $\bullet t \cup \{t\} \cup t^\bullet$ is in bijection with $\bullet t' \cup \{t'\} \cup t'^\bullet$ through φ .

Now, for N a safe net, there exists pairs (U, φ) where U is an occurrence net and $\varphi : U \mapsto N$ is a morphism — the places and transitions of U are “labeled” with places and transitions of N through morphism φ . The *unfolding* of N , denoted by (U_N, φ_N) or U_N for short, is the smallest pair (U, φ) with the above properties, where smallest refers to inclusion up to isomorphism. The configurations of U_N are the executions of N , seen as partial orders of events.

Figure 5 shows a safe net N and a small part of its unfolding U_N . A step in the construction of U_N is shown in the figure on the right. In each such step, a set of concurrent conditions X in U_N are chosen such that $\varphi(X) = \bullet t$ for some transition t in N . The set of nodes $X \cup \{t'\} \cup t'^\bullet$ are then added to U_N where $\varphi(t') = t$ and $\varphi(t'^\bullet) = t^\bullet$. The right figure shows the addition of a new copy of transition a (along with its preset and postset) to U_N .

Any place of an occurrence net (specifically, an unfolding) gets a token *at-most* once. We can thus talk about “the token of the place” unambiguously for any place in these nets.

Figure 5: Construction of the unfolding U_N of a safe net N .

3.2 Our Model: OrchNets

We now present the orchestration model that we use for our studies, which we call *OrchNets*. OrchNets are occurrence nets in which tokens are equipped with attributes (or colors). Figure 6 shows an OrchNet equipped with attributes

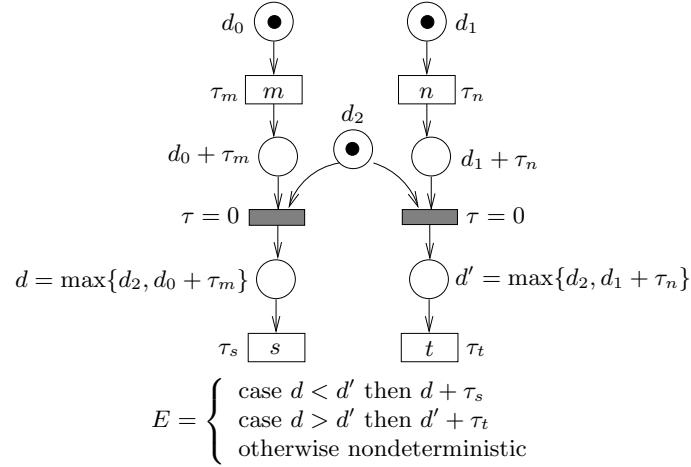


Figure 6: A Generic Transition of our Model. The arc expressions are shown next to the arcs and the guard expression is written next to the transition.

related to timing. Places are labeled with dates which is in fact the date of the token of that place. Transitions are labeled with latencies. The tokens in the three minimal places are given *initial dates* (here, d_0, d_1, d_2). The four named transitions m, n, s and t are labeled with *latencies* τ_m, τ_n, τ_s and τ_t respectively, and the two shaded transitions have zero latency.

The presence of dates in tokens alters the firing semantics slightly. A transition t is enabled at date d when all places in its preset have tokens. It then takes τ_t additional time to fire. For example, the shaded transition in the left has all its input tokens at $\max\{d_2, d_0 + \tau_m\}$ and so it fires at $\max\{d_2, d_0 + \tau_m\} + 0$ since it has zero latency. If a transition fires at date d , then the tokens in its postset have the date d . This is shown in the figure, e.g., on the place following the left shaded transition, which has date $\max\{d_2, d_0 + \tau_m\}$.

When transitions are in conflict, (e.g., the two shaded transitions in Figure 6), the transition that actually occurs is governed by a *race policy* [6]. If a set of enabled transitions are in conflict, the one with smallest date of occurrence will fire, preempting the other transitions in conflict with it. So, in Figure 6, the left shaded transition or the right shaded transition will fire depending on whether $d < d'$ or $d > d'$ respectively, with a nondeterministic choice if $d = d'$. This results in selecting the left most or right most continuation (firing s or t) depending on the above cases. The resulting overall latency E of the orchestration is shown at the bottom of the figure.

In addition to dates, tokens in OrchNets can have data attributes, which we call *values*. We have not shown this in Figure 6, in order to keep it simple. Values of tokens in the preset of a transition t can be combined by a value function ϕ_t attached to t . The resulting value is taken by the token in the postset of t .

Transitions can also have *guards* (the presence of such a guard is not necessary). Guards are boolean predicates involving values of the tokens in the transition's preset. A transition is enabled only if its guard is true. An “If(cond)-then-else” branching can be implemented by using the following mechanism: Put a place p , followed by two transitions t, t' . Guard t with the predicate “cond=true” and t' with the predicate “cond=false”.

At this point we are ready to provide the formal definition of *OrchNets*:

Definition 6 (OrchNet) *An OrchNet is a tuple $\mathcal{N} = (N, \Phi, T, T_{\text{init}})$ consisting of*

- *An occurrence net N with token attributes $c = (\text{value}, \text{date})$.*
- *A family $\Phi = (\phi_t)_{t \in \mathcal{T}}$ of value functions, whose input parameters are the values of the transition's input tokens.*
- *A family $T = (\tau_t)_{t \in \mathcal{T}}$ of latency functions, whose input parameters are the values of the transition's input tokens. The range of the latency functions is in $\mathbf{R} \cup \{+\infty\}$*
- *A family $T_{\text{init}} = (\tau_p)_{p \in \min(\mathcal{P})}$ of initial date functions for the minimal places of N . The range of the initial date functions is in $\mathbf{R} \cup \{+\infty\}$*

We do not specify a concrete range of value functions since they can be any arbitrary value and this is not of direct importance in the sequel. How a transition t modifies the attributes of tokens is formalized now: Let the preset of

t have n places whose tokens have $(value, date)$ attributes $(v_1, d_1) \dots (v_n, d_n)$. Then all the tokens in the postset of t have the pair (v_t, d_t) of value and date, where:

$$\begin{aligned} v_t &= \phi_t(v_1 \dots v_n) \\ d_t &= \max\{d_1 \dots d_n\} + \tau_t(v_1 \dots v_n) \end{aligned} \quad (1)$$

The *race policy* during execution is formalized as follows: In any given marking M , let T be the set of transitions that are *possibly enabled*, i.e. $\forall t \in T, \bullet t$ is marked in M . Then the transition t that is *actually enabled*, (which really fires) is given by:

$$t = \arg \min_{t \in T} d_t$$

where $\arg \min_{x \in X} f(x) = x^* \in X$ s.t. $\forall x' \in X, f(x^*) \leq f(x')$.

If two transitions have the same d_t , then the choice of the transition that actually fires is non-deterministic. If for a transition t , the delay d_t is infinite, it is equivalent to saying that transition t does not occur.

The choice extensions to occurrence nets in OrchNets is inspired by the application domain: compositions of web services. It reflects the following facts.

- Since we focus on latency, $(value, date)$ is the only needed color.
- Orchestrations rarely involve decisions on actions based on absolute dates. Timeouts are an exception, but these can be modelled explicitly, without using dates in guards of transitions. This justifies the fact that guards only have token values as inputs, and not their dates.
- The time needed to perform transitions does not depend on the tuple of dates $(d_1 \dots d_n)$ when input tokens were created, but it can depend on the data $(v_1 \dots v_n)$ and computation ϕ performed on these. This justifies our restriction for output arc expressions.

If it is still wished that control explicitly depends on dates, then dates must be measured and can then be stored as part of the value v .

Actually Occurring Configuration and Execution Time In general, both value and latency functions can be nondeterministic. We introduce an invisible daemon variable ω that resolves this nondeterminism and we denote by Ω its domain. For a given value of ω , the value and latency functions ϕ_t^ω and τ_t^ω are deterministic functions.

Let $\mathcal{N} = (N, \Phi, T, T_{\text{init}})$ be a finite OrchNet. For a value $\omega \in \Omega$ for the daemon we can calculate the following *dates* for every transition t and place p of \mathcal{N} :

$$\begin{aligned} d_p(\omega) &= \begin{cases} \tau_p^\omega & \text{if } p \text{ is minimal} \\ d_s(\omega) & \text{where } s = \bullet p \text{ otherwise} \end{cases} \\ d_t(\omega) &= \max\{d_n(\omega) \mid n \in \bullet t\} + \tau_t^\omega(v_1, \dots, v_n) \end{aligned} \quad (2)$$

where v_1, \dots, v_n are the value components of the tokens in $\bullet t$ as in equation (1). If κ is a configuration of N , the future \mathcal{N}^κ is the OrchNet $(N^\kappa, \Phi_{N^\kappa}, T_{N^\kappa}, T'_{\text{init}})$ where Φ_{N^κ} and T_{N^κ} are the restrictions of Φ and T respectively, to the transitions of N^κ . T'_{init} is the family derived from \mathcal{N} according to (2): for any minimal

place p of N^κ , the initialisation function is given by $\tau_p'^\omega = d_p(\omega)$. For X a set of nodes of N , let

$$\mathcal{T}_{\min}(X) = \{t \in \mathcal{T}(X) \mid \bullet\bullet t \cap X = \emptyset\}$$

Now define inductively,

$$\begin{aligned} \kappa_0(\omega) &= \emptyset \\ \kappa_m(\omega) &= \kappa_{m-1}(\omega) \cup \{t_m\} \cup \bullet t_m \cup t_m \bullet \\ &\text{where } t_m = \arg \min_{t \in \mathcal{T}_{\min}(N^{\kappa_{m-1}}(\omega))} d_t(\omega) \end{aligned} \quad (3)$$

Since net N is finite, the above inductive definition terminates in finitely many steps when $N^{\kappa_m(\omega)} = \emptyset$. Let $M(\omega)$ be this number of steps. We thus have

$$\emptyset = \kappa_0 \subset \kappa_1(\omega) \subset \dots \subset \kappa_{M(\omega)}(\omega)$$

$\kappa_{M(\omega)}(\omega)$ is a maximal configuration which actually occurs according to our timed semantics, for a fixed ω . Each step decreases the number of maximal configurations sharing $\kappa_m(\omega)$ as a prefix. We denote by $\bar{\kappa}(\mathcal{N}, \omega)$, the maximal configuration $\kappa_{M(\omega)}(\omega)$ that actually occurs.

For a prefix B of N define

$$E_\omega(B, \mathcal{N}) = \max\{d_x(\omega) \mid x \in B\} \quad (4)$$

If B is a configuration, then $E_\omega(B, \mathcal{N})$ is the time taken for B to execute (latency of B). The latency of the OrchNet $\mathcal{N} = (N, \Phi, T, T_{\text{init}})$, for a given ω is

$$E_\omega(\mathcal{N}) = E_\omega(\bar{\kappa}(\mathcal{N}, \omega), \mathcal{N})$$

4 Characterizing monotony

In this article, we are interested in the total time taken to execute a web-service orchestration. As a consequence, we will consider only orchestrations that terminate in a finite time, *i.e.*, only a finite number of values can be returned.

4.1 Defining monotony

To formalize monotony we must specify how latencies and initial dates can vary. As an example, we may want to constrain some pair of transitions to have identical latencies, or some pair of minimal places to have identical initial dates. This allowed flexibility in setting latencies or initial dates is formalized under the notion of pre-OrchNet we introduce next.

Definition 7 (pre-OrchNet) *Call pre-OrchNet a tuple $\mathbb{N} = (N, \Phi, \mathbb{T}, \mathbb{T}_{\text{init}})$, where N and Φ are as before, and \mathbb{T} and \mathbb{T}_{init} are sets of families T of latency functions and of families T_{init} of initial date functions. Write $\mathcal{N} \in \mathbb{N}$ if $\mathcal{N} = (N, \Phi, T, T_{\text{init}})$ for some $T \in \mathbb{T}$ and $T_{\text{init}} \in \mathbb{T}_{\text{init}}$.*

For two families T and T' of latency functions, write

$$T \geq T'$$

to mean that $\forall \omega \in \Omega, \forall t \in \mathcal{T} \implies \tau_t(\omega) \geq \tau'_t(\omega)$, and similarly for $T_{\text{init}} \geq T'_{\text{init}}$. For $\mathcal{N}, \mathcal{N}' \in \mathbb{N}$, write

$$\mathcal{N} \geq \mathcal{N}'$$

if $T \geq T'$ and $T_{\text{init}} \geq T'_{\text{init}}$ both hold.

Definition 8 (monotony) *pre-OrchNet* $\mathbb{N} = (N, \Phi, \mathbb{T}, \mathbb{T}_{\text{init}})$ is called monotonic if, for any two $\mathcal{N}, \mathcal{N}' \in \mathbb{N}$, such that $\mathcal{N} \geq \mathcal{N}'$, we have $E_\omega(\mathcal{N}) \geq E_\omega(\mathcal{N}')$.

Considering legal sets \mathbb{T} and \mathbb{T}_{init} of families of latency functions and initial date functions allows setting constraints on these families. This possibility will be essential in characterizing monotonic orchestrations.

4.2 A global necessary and sufficient condition

Theorem 1

1. The following condition implies the monotony of pre-OrchNet $\mathbb{N} = (N, \Phi, \mathbb{T}, \mathbb{T}_{\text{init}})$:

$$\begin{aligned} \forall \mathcal{N} \in \mathbb{N}, \quad \forall \omega \in \Omega, \forall \bar{\kappa} \in \bar{\mathcal{V}}(N), \\ E_\omega(\bar{\kappa}, \mathcal{N}) \geq E_\omega(\bar{\kappa}(\mathcal{N}, \omega), \mathcal{N}) \end{aligned} \quad (5)$$

where $\bar{\mathcal{V}}(N)$ denotes the set of all maximal configurations of net N and $\bar{\kappa}(\mathcal{N}, \omega)$ is the maximal configuration of \mathcal{N} that actually occurs under the daemon value ω .

2. Conversely, assume that:

- (a) Condition (5) is violated, and
- (b) for any two OrchNets \mathcal{N} and \mathcal{N}' such that $\mathcal{N} \in \mathbb{N}$, then $\mathcal{N}' \geq \mathcal{N} \implies \mathcal{N}' \in \mathbb{N}$.

Then $\mathbb{N} = (N, \Phi, \mathbb{T}, \mathbb{T}_{\text{init}})$ is not monotonic.

Sub-condition b) in statement 2 destroys the necessity of Condition (5). Statement 2 expresses that Condition (5) is also necessary provided that it is legal to increase at will latencies or initial dates. Observe that Condition (5) by itself cannot be enough since it trivially holds if \mathbb{T} is a singleton.

Proof: We first prove Statement 1. Let $\mathcal{N}' \in \mathbb{N}$ be such that $\mathcal{N}' \geq \mathcal{N}$. We have:

$$\begin{aligned} E_\omega(\bar{\kappa}(\mathcal{N}', \omega), \mathcal{N}') &\geq E_\omega(\bar{\kappa}(\mathcal{N}', \omega), \mathcal{N}) \\ &\geq E_\omega(\bar{\kappa}(\mathcal{N}, \omega), \mathcal{N}) \end{aligned}$$

where the first inequality follows from the fact that $\bar{\kappa}(\mathcal{N}', \omega)$ is a conflict free partial order and $\mathcal{N}' \geq \mathcal{N}$, and the second inequality follows from (5) applied with $\bar{\kappa} = \bar{\kappa}(\mathcal{N}', \omega)$. This proves Statement 1.

We prove statement 2 by contradiction. Let $(\mathcal{N}, \omega, \bar{\kappa}^\dagger)$ be a triple violating Condition (5), in that

$$\bar{\kappa}^\dagger \text{ cannot occur} \quad (6)$$

$$E_\omega(\bar{\kappa}^\dagger, \mathcal{N}) < E_\omega(\bar{\kappa}(\mathcal{N}, \omega), \mathcal{N}) \quad (7)$$

Now consider the OrchNet net $\mathcal{N}' = (N, \Phi, T', T_{\text{init}})$ where the family T' is the same as T except that in ω , $\forall t \notin \bar{\kappa}^\dagger$, $\tau'_t(\omega) > E_\omega(\bar{\kappa}^\dagger, \mathcal{N})$. Clearly $\mathcal{N}' \geq \mathcal{N}$. But using construction (3), it is easy to verify that $\bar{\kappa}(\mathcal{N}', \omega) = \bar{\kappa}^\dagger$ and thus

$$\begin{aligned} E_\omega(\bar{\kappa}(\mathcal{N}', \omega), \mathcal{N}') &= E_\omega(\bar{\kappa}^\dagger, \mathcal{N}') \\ &= E_\omega(\bar{\kappa}^\dagger, \mathcal{N}) \\ &< E_\omega(\bar{\kappa}(\mathcal{N}, \omega), \mathcal{N}), \end{aligned}$$

which violates monotony. \diamond

4.3 A structural condition for the monotony of workflow nets

Workflow nets [11] were proposed as a simple model for workflows. These are Petri nets, with a special minimal place i and a special maximal place o . We consider the class of workflow nets that are 1-safe and which have no loops. Further, we require them to be *sound* [11].

Definition 9 *A Workflow net W is said to be sound iff:*

1. *For every marking M reachable from the initial place i , there is a firing sequence leading to the final place o .*
2. *If a marking M marks the final place o , then no other place can in W can be marked in M*
3. *There are no dead transitions in W . Starting from the initial place, it is always possible to fire any transition of W .*

An example of workflow net is shown in the first net of Figure 4. Workflow nets will be generically denoted by W . We can equip workflow nets with same attributes as occurrence nets, this yields *pre-WFnets* $\mathbb{W} = (W, \Phi, \mathbb{T}, \mathbb{T}_{\text{init}})$. Referring to the end of Section 3.1, unfolding W yields an occurrence net that we denote by N_W with associated morphism $\varphi_W : N_W \mapsto W$, see Figure 4. Here the morphism φ_W maps the two c transitions (and the place in its preset and postset) in the net on the right to the single c transition (and its preset and postset) in the net on the left. Observe that W and N_W possess identical sets of minimal places. Morphism φ_W induces a pre-OrchNet

$$\mathbb{N}_W = (N_W, \Phi_W, \mathbb{T}_W, \mathbb{T}_{\text{init}})$$

by attaching to each transition t of N_W the value and latency functions attached to $\varphi_W(t)$ in \mathbb{W} .

We shall use the results of the previous section in order to characterize those pre-WFnets whose unfoldings give monotonic pre-OrchNets. Our characterization will be essentially structural in that it does not involve any constraint on latency functions beyond equality constraints, for some pairs of transitions. Under this restricted discipline, the simple structural conditions we shall formulate will also be almost necessary.

We first define a notion of *cluster* on safe nets, which will be useful for our characterisation.

Definition 10 (clusters) For a safe net N , a cluster is a minimal set \mathbf{c} of places and transitions of N such that

$$\forall t \in \mathbf{c} \implies \bullet t \subseteq \mathbf{c} \quad , \quad \forall p \in \mathbf{c} \implies p^\bullet \subseteq \mathbf{c} \quad (8)$$

Theorem 2 (Sufficient Condition) Let W be a WFnet and N_W be its unfolding. A sufficient condition for the pre-OrchNet $\mathbb{N}_W = (N_W, \Phi_W, \mathbb{T}_W, \mathbb{T}_{\text{init}})$ to be monotonic is that every cluster \mathbf{c} satisfies the following condition:

$$\forall t_1, t_2 \in \mathbf{c}, t_1 \neq t_2 \implies t_1^\bullet = t_2^\bullet \quad (9)$$

Proof: Let φ_W be the net morphism mapping N_W onto W and let $\mathcal{N} \in \mathbb{N}$ be any OrchNet. We prove that condition 1 of Theorem 1 holds for \mathcal{N} by induction on the number of transitions in the maximal configuration $\bar{\kappa}(\mathcal{N}, \omega)$ that actually occurs. The base case is when it has only one transition. Clearly this transition has the least latency and any other maximal configuration has a greater execution time.

Induction Hypothesis Condition 1 of Theorem 1 holds for any maximal occurring configuration with $m - 1$ transitions ($m > 1$). Formally, for a pre-OrchNet $\mathbb{N} = (N, \Phi, \mathbb{T}, \mathbb{T}_{\text{init}})$: $\forall \mathcal{N} \in \mathbb{N}, \forall \omega \in \Omega, \forall \bar{\kappa} \in \bar{\mathcal{V}}(N)$,

$$E_\omega(\bar{\kappa}, \mathcal{N}) \geq E_\omega(\bar{\kappa}(\mathcal{N}, \omega), \mathcal{N}) \quad (10)$$

holds if $|\{t \in \bar{\kappa}(\mathcal{N}, \omega)\}| \leq m - 1$.

Induction Argument Consider the OrchNet \mathcal{N} , where the actually occurring configuration $\bar{\kappa}(\mathcal{N}, \omega)$ has m transitions. κ' is any other maximal configuration of \mathcal{N} . If the transition t in $\bar{\kappa}(\mathcal{N}, \omega)$ with minimal date d_t also occurs in κ' then comparing execution times of $\bar{\kappa}(\mathcal{N}, \omega)$ and κ' reduces to comparing $E_\omega(\bar{\kappa}(\mathcal{N}, \omega) \setminus \{t\}, \mathcal{N}^t)$ and $E_\omega(\kappa' \setminus \{t\}, \mathcal{N}^t)$. Since $\bar{\kappa}(\mathcal{N}, \omega) \setminus \{t\}$ is the actually occurring configuration in the future \mathcal{N}^t of transition t , using our induction hypothesis, we have

$$E_\omega(\bar{\kappa}(\mathcal{N}, \omega) \setminus \{t\}, \mathcal{N}^t) \leq E_\omega(\kappa' \setminus \{t\}, \mathcal{N}^t)$$

and so

$$E_\omega(\bar{\kappa}(\mathcal{N}, \omega), \mathcal{N}) \leq E_\omega(\kappa', \mathcal{N})$$

If $t \notin \kappa'$ for some κ' , then there must exist another transition t' such that $\bullet t \cap \bullet t' \neq \emptyset$. By the definition of clusters, $\varphi_W(t)$ and $\varphi_W(t')$ must belong to the same cluster \mathbf{c} . Hence, $t^\bullet = t'^\bullet$ follows from condition 9 of Theorem 2. The futures \mathcal{N}^t and $\mathcal{N}^{t'}$ thus have identical sets of transitions: they only differ in the initial marking of their places. If T_{init} and T'_{init} are the initial marking of these places, $T_{\text{init}} \leq T'_{\text{init}}$ (since $d_t \leq d_{t'}$, t^\bullet has dates lesser than t'^\bullet). Hence

$$E_\omega(\bar{\kappa}(\mathcal{N}, \omega), \mathcal{N}) = E_\omega(\bar{\kappa}(\mathcal{N}, \omega) \setminus \{t\}, \mathcal{N}^t) \quad (11)$$

and

$$\begin{aligned} E_\omega(\kappa', \mathcal{N}) &= E_\omega(\kappa' \setminus \{t'\}, \mathcal{N}^{t'}) \\ &\geq E_\omega(\kappa' \setminus \{t'\}, \mathcal{N}^t) \end{aligned} \quad (12)$$

The inequality holds since $\mathcal{N}^{t'} \geq \mathcal{N}^t$. The induction hypothesis on (11) and (12) gives

$$E_\omega(\overline{\kappa}(\mathcal{N}, \omega), \mathcal{N}) \leq E_\omega(\kappa', \mathcal{N})$$

This proves the theorem. \diamond

Comments about tightness of the conditions of Theorem 2 Recall that the sufficient condition for monotony stated in Theorem 1 is “almost necessary” in that, if enough flexibility exist in setting latencies and initial dates, then it is actually necessary. It turns out that the same holds for the sufficient condition stated in Theorem 2 if the workflow net is assumed to be live.

Theorem 3 (Necessary Condition) *Suppose that the workflow net W is sound. Assume that $\mathcal{W} \in \mathbb{W}$ and $\mathcal{W}' \geq \mathcal{W}$ implies $\mathcal{W}' \in \mathbb{W}$, meaning that there is enough flexibility in setting latencies and initial dates. In addition, assume that there is atleast one $\mathcal{W}^* \in \mathbb{W}$ such that there is an daemon value ω^* for which the latencies of all the transitions are finite. Then the sufficient condition of Theorem 2 is also necessary for monotony.*

Proof: We will show that when condition (9) of Theorem 2 is not satisfied by W , the Orchnets in its induced preOrchNet \mathbb{N}_W can violate condition (5) of Theorem 1, the necessary condition for monotony.

Let c_W be any cluster in W that violates the condition 9 of Theorem 2. Consider the unfolding of W , N_W and the associated morphism $\varphi : N_W \mapsto W$ as introduced before. Since W is sound, all transitions in c_W are reachable from the initial place i and so there is a cluster c in N_W such that $\varphi(c) = c_W$. There are transitions $t_1, t_2 \in c$ such that $\bullet t_1 \cap \bullet t_2 \neq \emptyset$, $\bullet \varphi(t_1) \cap \bullet \varphi(t_2) \neq \emptyset$ and $\varphi(t_1)^\bullet \neq \varphi(t_2)^\bullet$. Call $[t] = \lceil t \rceil \setminus \{t\}$ and define $K = [t_1] \cup [t_2]$. We consider the following two cases:

K is a configuration If so, consider the OrchNet $\mathcal{N}^* \in \mathbb{N}_W$ got when transitions of N_W (and so W) have latencies as that in \mathcal{W}^* . So for the daemon value ω^* , the quantity $E_{\omega^*}(K, \mathcal{N}^*)$ is some finite value \mathbf{n}^* . Now, configuration K can actually occur in a OrchNet \mathcal{N} , such that $\mathcal{N} > \mathcal{N}^*$, where \mathcal{N} is obtained as follows (τ and τ^* denote the latencies of transitions in \mathcal{N} and \mathcal{N}^* respectively): $\forall t \in K, t' \in N_W$ s.t. $\bullet t \cap \bullet t' \neq \emptyset$, set $\tau_{t'}(\omega^*) = \mathbf{n}^* + 1$ and keep the other latencies unchanged. In this case, for the daemon value ω^* , the latencies of all transitions of \mathcal{N} (and so its overall execution time) is finite. Denote by \mathcal{N}^K the future of \mathcal{N} once configuration K has actually occurred. Both t_1 and t_2 are minimal and enabled in \mathcal{N}^K .

Since $\varphi(t_1)^\bullet \neq \varphi(t_2)^\bullet$, without loss of generality, we assume that there is a place $p \in t_1^\bullet$ such that $\varphi(p) \in \varphi(t_1)^\bullet$ but $\varphi(p) \notin \varphi(t_2)^\bullet$. Let t^* be a transition in \mathcal{N}^K such that $t^* \in p^\bullet$. Such a transition must exist since p can not be a maximal place: $\varphi(p)$ can not be a maximal place in W which has a unique maximal place. Now consider the Orchnet $\mathcal{N}' > \mathcal{N}$ got as follows: $\tau'_{t_1}(\omega^*) = \tau_{t_1}(\omega^*)$, $\tau'_{t_2}(\omega^*) = \tau_{t_2}(\omega^*) + 1$ and for all other $t \in c$, $\tau'_t(\omega^*) = \tau_t(\omega^*) + 1$. Set $\tau'_{t^*}(\omega^*) = \infty$ and for all other transitions of \mathcal{N}' , the delays are the same as that in \mathcal{N} and thus are finite for ω^* .

t_1 has the minimal delay among all transitions in c , and t^* is in the future of t_1 . So the actually occurring configuration $E_{\omega^*}(\bar{\kappa}(\mathcal{N}', \omega^*), \mathcal{N}')$ has an infinite delay. However any maximal configuration $\bar{\kappa}$ which does not include t_1 (for eg, when t_2 fires instead of t_1) will have a finite delay. For such $\bar{\kappa}$ we thus have $E_{\omega^*}(\bar{\kappa}(\mathcal{N}', \omega^*), \mathcal{N}') > E_{\omega^*}(\bar{\kappa}, \mathcal{N}')$ and so \mathcal{N}' violates the condition (5) of Theorem 1.

K is not a configuration If so, there exist transitions $t \in [t_1] \setminus [t_2]$, $t' \in [t_2] \setminus [t_1]$ such that $\bullet t \cap \bullet t' \neq \emptyset$, $\bullet \varphi(t) \cap \bullet \varphi(t') \neq \emptyset$ and $\varphi(t)^\bullet \neq \varphi(t')^\bullet$. The final condition holds since t_2 and t_1 are not in the causal future of t and t' respectively. Thus t and t' belong to the same cluster, which violates condition 9 of Theorem 2 and we can apply the same reasoning as in the beginning of the proof. Since $[t]$ is finite for any transition t , we will eventually end up with K being a configuration. \diamond

4.4 Discussion regarding monotony

Based on the mathematical results of the former section, we shall first discuss which orchestrations are monotonic. The resulting class is not very wide, as we shall see. However, further thinking suggests that considering QoS parameters in isolation — as we did so far — is not the right way to proceed. We will thus revisit QoS and monotony.

Which orchestrations are monotonic? Not surprisingly, orchestrations involving no choice at all — they are modeled by event graphs — are monotonic.

Theorem 2, however, allows for more general orchestrations. In particular, if multi-threading with only conflicting threads are used, then the theorem essentially requires that the choice among the different threads is performed, based on their overall latency (where a thread is considered as atomic). It is allowed, however, that conflicting threads possess different overall latencies. Next, if a blend of concurrent and conflicting multi-threading is used, then 1/ concurrent threads should have equal latencies, and 2/ the choice among conflicting threads should be based on their overall latency. Finally, concurrent multi-threading (with no conflict with other thread when the concurrent threads are launched) raises no problem. An example of a non trivial monotonic orchestration is shown in Figure 4.

Revisiting QoS and monotony As discussed before, orchestrations involving data dependent control will very often be non monotonic. This makes SLA/SLS/contract based QoS management very problematic. What did we wrong?

In fact, the problem is that we consider QoS parameters in isolation when dealing with SLS or contracts. However one can trade latency for “quality of data”. Typically, by reducing timeouts while waiting for responses from called Web services, one can easily reduce latency. But there is no free lunch: if one does so, then exceptions get raised in lieu of getting valid responses for the query.

So we must refine our notion of monotony as follows: say that an orchestration is “conditionally monotonic” if, *under the constraint that identical responses*

are received by the orchestration, increasing some latency will cause an increase of the overall latency of the orchestration. In the next section we formalize this notion and study the characterization of conditional monotony.

5 Refined QoS and Conditional monotony

Define the set of values returned by a maximal configuration $\kappa \in \overline{\mathcal{V}}(\mathcal{N})$ as

$$V_\omega(\kappa, \mathcal{N}) = \{\phi_t(\omega) \mid t \in \max(\kappa)\}$$

where $\max(\kappa)$ denotes the maximal (w.r.t \leq) transitions in the configuration κ . The values returned by the orchestration $\mathcal{N} = (N, \phi, T, T_{\text{init}})$ for a given value of the daemon ω is

$$V_\omega(\mathcal{N}) = V_\omega(\overline{\kappa}(\mathcal{N}, \omega), \mathcal{N})$$

Definition 11 (conditional monotony) *pre-OrchNet $\mathbb{N} = (N, \Phi, \mathbb{T}, \mathbb{T}_{\text{init}})$ is called conditionally monotonic if,*

$$\left. \begin{array}{l} \forall \mathcal{N}, \mathcal{N}' \in \mathbb{N} \text{ s. t. } \mathcal{N} \geq \mathcal{N}' \text{ and} \\ \forall \omega \in \Omega \text{ s. t. } V_\omega(\mathcal{N}) = V_\omega(\mathcal{N}') \end{array} \right\} \Rightarrow E_\omega(\mathcal{N}) \geq E_\omega(\mathcal{N}')$$

Definition 11 does not attempt to compare overall orchestration latencies, if different responses are received as a result of changing latencies of called Web services. In particular, cases where valid data are received are not compared with cases where exceptions are raised. Said differently, Definition 11 prohibits trading latency for quality of data.

Conditional monotony does not seem to be considered while formulating WSLA contracts. The 'conditional' contracts in WSLA are not related to the notion of conditional monotony but rather refer to the obligations that need to be met by the client for the server's promises to be fulfilled.

Assumption 1 *We assume that OrchNet N is such that for all distinct $\kappa, \kappa' \in \overline{\mathcal{V}}(\mathcal{N})$, $V_\omega(\kappa, \mathcal{N}) \neq V_\omega(\kappa', \mathcal{N})$.*

Assumption 1 is very natural when normal processing of the orchestration together with cases where exceptions occur are considered. With this assumption, Theorem 1 simplifies as follows:

Theorem 4 *Under Assumption 1 pre-OrchNet $\mathbb{N} = (N, \Phi, \mathbb{T}, \mathbb{T}_{\text{init}})$ is conditionally monotonic.*

Proof: In fact this is a trivial result: by Assumption 1, we do not need to compare latencies across different configurations. But each configuration possesses no conflict and is therefore an event graph. Since dating in event graphs is amenable of $\max/+$ algebra, it is monotonic. Thus pre-OrchNet $\mathbb{N} = (N, \Phi, \mathbb{T}, \mathbb{T}_{\text{init}})$ is conditionally monotonic. \diamond

6 Conclusion

This paper is a contribution to the fundamentals of contract based QoS management of Web services orchestrations/choreographies. QoS contracts implicitly assume monotony w.r.t. QoS parameters. This paper focuses on one specific (but representative) QoS parameter, namely latency or response time. We have shown that monotony is very easily violated in realistic cases. We have formalized monotony and have provided necessary and sufficient conditions for it. As we have seen, QoS can be very often traded for Quality of Data: poor quality responses to queries (including exceptions or invalid responses) can be obtained typically much faster. This reveals that QoS parameters should not be considered separately, in isolation. We have thus revisited the notions of latency and monotony and proposed the new concept of *conditional monotony*. Fortunately enough, typical orchestrations turn out to be non-monotonic, but conditionally monotonic.

Our study has an impact on the way SLS should be phrased for orchestrations: it suggests crude contracts such as “for 95% of the requests the response time will be less than 5ms” should not be used without referring to the quality of data. Also, our study has an impact on contract monitoring — monitoring whether a called service meets its contract.

We see two extensions of this work. First, our mathematical results rely on the notion of branching cells, which were developed for nets without read arcs. However, advanced orchestration languages such as Orc [7] offer a sophisticated form of preemption that requires contextual nets (with read arcs) for their modeling. Extending our results to this case is non trivial since branching cells do not exist for contextual nets. Second, this paper considers latencies in a non probabilistic context. However, these authors advocated in [9] the use of probabilistic contracts — they better reflect the uncertain nature of QoS parameters in the open world of the Web and they allow for well sound overbooking and less pessimistic contracts. We should extend our present work to this probabilistic framework. This should not be too difficult, as we guess. In fact, a theory of monotony for probabilistic QoS must rely on an order among probability distributions. This exists and is known as *stochastic ordering*. Results are available that relate stochastic ordering and point-wise ordering (as we study here), which should be of help for such an extension.

References

- [1] W.M.P. van der Aalst. The Application of Petri Nets to Workflow Management. *The Journal of Circuits, Systems and Computers*, 8(1):21–66, 1998.
- [2] Tony Andrews, Francisco Curbera, Hitesh Dholakia, Yaron Goland, Johannes Klein, Frank Leymann, Kevin Liu, Dieter Roller, Doug Smith, Satish Thatte, Ivana Trickovic, Sanjiva Weerawarana, and Satish Thatte. Business Process Execution Language for Web Services Version 1.1. Specification, BEA Systems, International Business Machines Corporation, Microsoft Corporation, SAP AG, Siebel Systems, May 2003.

- [3] Jesús Arias-Fisteus, Luis Sánchez Fernández, and Carlos Delgado Kloos. Applying model checking to BPEL4WS business collaborations. In *SAC*, pages 826–830, 2005.
- [4] Alexander Keller and Heiko Ludwig. The wsla framework: Specifying and monitoring service level agreements for web services. *J. Network Syst. Manage.*, 11(1), 2003.
- [5] David Kitchen, William R. Cook, and Jayadev Misra. A language for task orchestration and its semantic properties. In *CONCUR*, pages 477–491, 2006.
- [6] Marco Ajmone Marsan, Gianfranco Balbo, Gianni Conte, Susanna Donatelli, and Giuliana Franceschinis. Modelling with generalized stochastic petri nets. *SIGMETRICS Performance Evaluation Review*, 26(2):2, 1998.
- [7] Jayadev Misra and William R. Cook. Computation orchestration: A basis for wide-area computing. *Journal of Software and Systems Modeling*, May, 2006. Available for download at <http://dx.doi.org/10.1007/s10270-006-0012-1>.
- [8] C. Ouyang, E. Verbeek, W.M.P van der Aalst, and S. Breutel. Formal Semantics and Analysis of Control Flow in WS-BPEL. BPM Center Report BPM-05-15, BPMcenter.org, 2005.
- [9] Sidney Rosario, Albert Benveniste, Stefan Haar, and Claude Jard. Probabilistic QoS and soft contracts for transaction based web services. In *ICWS*, pages 126–133, 2007.
- [10] Sidney Rosario, David Kitchen, Albert Benveniste, William Cook, Stefan Haar, and Claude Jard. Event structure semantics of orc. In *WSFM*, 2007.
- [11] Wil M. P. van der Aalst. Verification of workflow nets. In *ICATPN*, pages 407–426, 1997.



Centre de recherche INRIA Rennes – Bretagne Atlantique
IRISA, Campus universitaire de Beaulieu - 35042 Rennes Cedex (France)

Centre de recherche INRIA Bordeaux – Sud Ouest : Domaine Universitaire - 351, cours de la Libération - 33405 Talence Cedex
Centre de recherche INRIA Grenoble – Rhône-Alpes : 655, avenue de l'Europe - 38334 Montbonnot Saint-Ismier
Centre de recherche INRIA Lille – Nord Europe : Parc Scientifique de la Haute Borne - 40, avenue Halley - 59650 Villeneuve d'Ascq
Centre de recherche INRIA Nancy – Grand Est : LORIA, Technopôle de Nancy-Brabois - Campus scientifique
615, rue du Jardin Botanique - BP 101 - 54602 Villers-lès-Nancy Cedex
Centre de recherche INRIA Paris – Rocquencourt : Domaine de Voluceau - Rocquencourt - BP 105 - 78153 Le Chesnay Cedex
Centre de recherche INRIA Saclay – Île-de-France : Parc Orsay Université - ZAC des Vignes : 4, rue Jacques Monod - 91893 Orsay Cedex
Centre de recherche INRIA Sophia Antipolis – Méditerranée : 2004, route des Lucioles - BP 93 - 06902 Sophia Antipolis Cedex

Éditeur
INRIA - Domaine de Voluceau - Rocquencourt, BP 105 - 78153 Le Chesnay Cedex (France)
<http://www.inria.fr>
ISSN 0249-6399